PATENT ABSTRACTS OF JAPAN

(11)Publication number : 2001-022591
(43)Date of publication of application : 26.01.2001

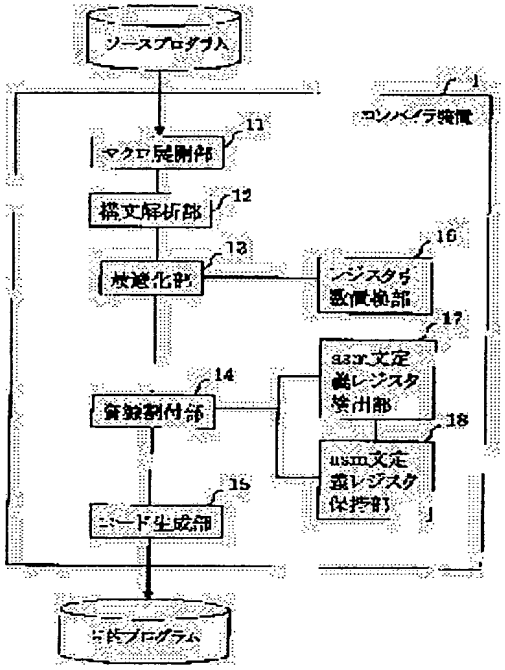
(51)Int.Cl. G06F 9/45

(21)Application number : 11-195717
(22)Date of filing : 09.07.1999

(71)Applicant : MATSUSHITA ELECTRIC IND CO LTD
(72)Inventor : TANAKA AKIRA

(54) COMPILER DEVICE, COMPUTER-READABLE RECORDING MEDIUM WHERE COMPILING PROGRAM IS RECORDED, AND COMPILING METHOD

(57)Abstract:
PROBLEM TO BE SOLVED: To translate a program without forcing a programmer to make a check in the use of an in-line assembly routine by allocating a resource different from a detected resource to an unallocated variable.
SOLUTION: A macro development part 11 replaces a macro-defined macro identifier with its main body. A syntax analysis part 12 analyzes and performs conversion into an intermediate program by performing the character and phrase interpretation, syntax analysis, and meaning analysis of the macro-developed program. Then an optimization part 13 is actuated to optimize the intermediate program for the reduction of the program size and processing execution time of an object program, and a resource allocation part 14 calculates the Life sections of all variables by using analyzed control flow and a data flow information. Then the compiling device 1 starts a code generating device 15 to convert intermediate instructions by a compiler to the object program of assembler sentences, machine word instructions, etc., of a target machine.



LEGAL STATUS

[Date of request for examination] 27.09.2004

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

[Date of registration]

[Number of appeal against examiner's decision of rejection]

[Date of requesting appeal against examiner's decision of rejection]

[Date of extinction of right]

(19)日本国特許庁（ＪＰ）　(12)公開特許公報（Ａ）　(11)特許出願公開番号
特開2001－22591
（P2001－22591A）
(43)公開日　平成13年１月26日(2001.1.26)

(51)Int.Cl.[7]	識別記号	Ｆ Ｉ	ﾃｰﾏｺｰﾄﾞ(参考)
Ｇ０６Ｆ　9/45		Ｇ０６Ｆ　9/44	３２２Ｈ　５Ｂ０８１

審査請求　未請求　請求項の数11　ＯＬ　（全 18 頁）

(21)出願番号　特願平11－195717

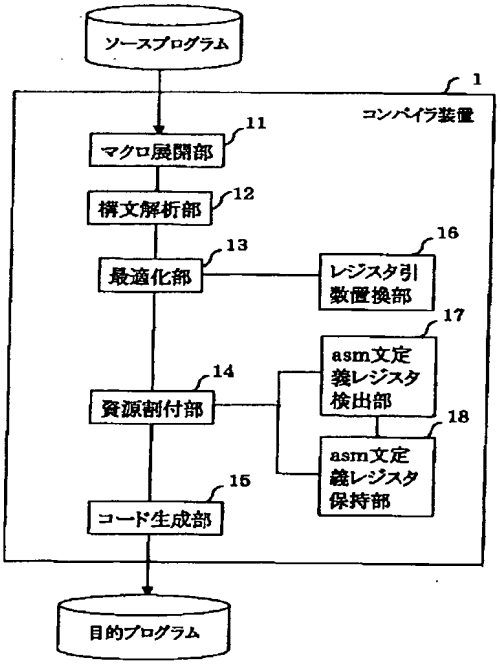
(22)出願日　平成11年７月９日(1999.7.9)

(71)出願人　000005821
松下電器産業株式会社
大阪府門真市大字門真1006番地
(72)発明者　田中　旭
大阪府門真市大字門真1006番地　松下電器産業株式会社内
(74)代理人　100090446
弁理士　中島　司朗　（外１名）

Ｆターム(参考)　5B081 AA06 AA07 CC25

(54)【発明の名称】　コンパイラ装置、コンパイルプログラムが記録されたコンピュータ読み取り可能な記録媒体及びコンパイル方法

(57)【要約】
【課題】　高級言語で書かれたプログラムを機械語プログラムに翻訳するコンパイラ装置において、インラインアセンブラルーチンの再利用性を向上させることを目的とする。
【解決手段】　レジスタで引き渡される仮引数に関しては、仮引数自体をコンパイラ内部で生成した変数に置き換え、さらに、変数へのレジスタ割り付け時に、変数の生存区間がインラインアセンブラルーチンを含む場合、インラインアセンブラルーチンで更新されるレジスタとは異なるレジスタを変数に割り付ける。



1

【特許請求の範囲】

【請求項１】　変数を定義又は参照している文を含むプログラムを、オブジェクト命令列に翻訳するコンパイラ装置であって、

前記オブジェクト命令列は、レジスタ又はメモリである資源を定義又は参照している命令からなり、

前記プログラムの何れかの部位は、何れかの資源を定義又は参照しているアセンブラ文により占有されており、

前記コンパイラ装置は、

どの資源に割り付けるべきかが未定である未割付変数のなかから、その生存区間がアセンブラ文の占有部位と重複しているものを検出する未割付変数検出手段と、

未割付変数検出手段により検出された未割付変数の生存区間と重複しているアセンブラ文にて定義されている資源を検出する資源検出手段と、

資源検出手段により検出された資源とは異なる資源を未割付変数検出手段により検出された未割付変数に割り付ける資源割付手段とを備えることを特徴とするコンパイラ装置。

【請求項２】　前記プログラムは、複数の関数を含み、

前記コンパイラ装置は更に、

何れかの関数にレジスタを用いて引渡される仮引数が存在する場合、当該仮引数の値を一時変数に代入する旨の代入命令を生成して当該関数の最初の位置に挿入すると共に、関数において使用されている全ての仮引数を、前記代入命令において値が代入される一時変数に置き換えるレジスタ引数置換手段を備え、

前記未割付変数検出手段は更に、

解析された生存区間内にアセンブラ文が含まれている一時変数を検出し、

前記資源検出手段は更に、

生存区間に含まれている当該アセンブラ文にて定義又は参照されているレジスタを検出して、

前記資源割付手段は、

未割付変数検出手段により検出された一時変数に、資源検出手段により検出されたレジスタとは異なるレジスタを割り付けることを特徴とする請求項１に記載のコンパイラ装置。

【請求項３】　前記未割付変数検出手段は、

前記プログラムからアセンブラ文を検出する検出部と、

アセンブラ文が検出されると、当該アセンブラ文にて定義又は参照される資源に割り付けられた割付済み一時変数を生成する割付済み一時変数生成部と、

前記プログラムにおいて検出されたアセンブラ文が占めている部位を検出し、検出された占有部位を生存区間として、前記生成された割付済み一時変数に設定する生存区間設定部と、

割付済み一時変数に設定された生存区間と、生存区間が重なる未割付変数を検出する未割付変数検出部とを備えることを特徴とする請求項１又は２に記載のコンパイラ

2

装置。

【請求項４】　前記コンパイラ装置は、

変数を定義又は参照している文と、マクロ識別子とを含むソースプログラムを装置外部から読み出す読出手段と、

マクロ識別子に基づいてソースプログラム内にアセンブラ文を展開するマクロ展開手段と、

展開されたアセンブラ文を含むソースプログラムの構文を解析して、複数の中間命令からなる中間プログラムを得る構文解析手段と、

構文解析により得られた中間プログラムに含まれる各変数がどの中間命令で定義され、どの中間命令で最後に参照されているか解析することにより、中間プログラムにおける未割付変数について生存区間を解析する生存区間解析手段とを備え、

前記未割付変数検出手段は、

解析された生存区間がアセンブラ文の占有部位と重複している未割付変数を検出することを特徴とする請求項１～３の何れかに記載のコンパイラ装置。

【請求項５】　変数を定義又は参照している文を含むプログラムを、オブジェクト命令列に翻訳するコンパイルプログラムを記録したコンピュータ読取可能な記録媒体であって、

前記プログラムは前記オブジェクト命令列は、レジスタ又はメモリである資源を定義又は参照している命令からなり、

前記プログラムの何れかの部位は、何れかの資源を定義又は参照しているアセンブラ文により占有されており、

前記コンパイルプログラムは、

どの資源に割り付けるべきかが未定である未割付変数のなかから、その生存区間がアセンブラ文の占有部位と重複しているものを検出する未割付変数検出ステップと、

未割付変数検出ステップにより検出された未割付変数の生存区間と重複しているアセンブラ文にて定義されている資源を検出する資源検出ステップと、

資源検出ステップにより検出された資源とは異なる資源を未割付変数検出ステップにより検出された未割付変数に割り付ける資源割付ステップとからなることを特徴とするコンピュータ読み取り可能な記録媒体。

【請求項６】　前記プログラムは、複数の関数を含み、

前記コンパイルプログラムは更に、

何れかの関数にレジスタを用いて引渡される仮引数が存在する場合、当該仮引数の値を一時変数に代入する旨の代入命令を生成して当該関数の最初の位置に挿入すると共に、関数において使用されている全ての仮引数を、前記代入命令において値が代入される一時変数に置き換えるレジスタ引数置換ステップを備え、

前記未割付変数検出ステップは更に、

解析された生存区間内にアセンブラ文が含まれている一時変数を検出し、

前記資源検出ステップは更に、
生存区間に含まれている当該アセンブラ文にて定義又は参照されているレジスタを検出して、
前記資源割付ステップは、
未割付変数検出ステップにより検出された一時変数に、資源検出ステップにより検出されたレジスタとは異なるレジスタを割り付けることを特徴とする請求項５に記載のコンピュータ読み取り可能な記録媒体。
【請求項７】　前記未割付変数検出ステップは、
前記プログラムからアセンブラ文を検出する検出サブステップと、
アセンブラ文が検出されると、当該アセンブラ文にて定義又は参照される資源に割り付けられた割付済み一時変数を生成する割付済み一時変数生成サブステップと、
前記プログラムにおいて検出されたアセンブラ文が占めている部位を検出し、検出された占有部位を生存区間として、前記生成された割付済み一時変数に設定する生存区間設定サブステップと、
割付済み一時変数に設定された生存区間と、生存区間が重なる未割付変数を検出する未割付変数検出サブステップとからなることを特徴とする請求項５又は６に記載のコンピュータ読み取り可能な記録媒体。
【請求項８】　前記コンパイルプログラムは、
変数を定義又は参照している文と、マクロ識別子とを含むソースプログラムを装置外部から読み出す読出ステップと、
マクロ識別子に基づいてソースプログラム内にアセンブラ文を展開するマクロ展開ステップと、
展開されたアセンブラ文を含むソースプログラムの構文を解析して、複数の中間命令からなる中間プログラムを得る構文解析ステップと、
構文解析により得られた中間プログラムに含まれる各変数がどの中間命令で定義され、どの中間命令で最後に参照されているか解析することにより、中間プログラムにおける未割付変数について生存区間を解析する生存区間解析ステップとを備え、
前記未割付変数検出ステップは、
解析された生存区間がアセンブラ文の占有部位と重複している未割付変数を検出することを特徴とする請求項５～７の何れかに記載のコンピュータ読み取り可能な記録媒体。
【請求項９】　変数を定義又は参照している文を含むプログラムを、オブジェクト命令列に翻訳するコンパイル方法であって、
前記プログラムは前記オブジェクト命令列は、レジスタ又はメモリである資源を定義又は参照している命令からなり、
前記プログラムの何れかの部位は、何れかの資源を定義又は参照しているアセンブラ文により占有されており、
前記コンパイルプログラムは、
どの資源に割り付けるべきかが未定である未割付変数のなかから、その生存区間がアセンブラ文の占有部位と重複しているものを検出する未割付変数検出ステップと、
未割付変数検出ステップにより検出された未割付変数の生存区間と重複しているアセンブラ文にて定義されている資源を検出する資源検出ステップと、
資源検出ステップにより検出された資源とは異なる資源を未割付変数検出ステップにより検出された未割付変数に割り付ける資源割付ステップとからなることを特徴とするコンパイル方法。
【請求項１０】　前記プログラムは、複数の関数を含み、
前記コンパイルプログラムは更に、
何れかの関数にレジスタを用いて引渡される仮引数が存在する場合、当該仮引数の値を一時変数に代入する旨の代入命令を生成して当該関数の最初の位置に挿入すると共に、関数において使用されている全ての仮引数を、前記代入命令において値が代入される一時変数に置き換えるレジスタ引数置換ステップを備え、
前記未割付変数検出ステップは更に、
解析された生存区間内にアセンブラ文が含まれている一時変数を検出し、
前記資源検出ステップは更に、
生存区間に含まれている当該アセンブラ文にて定義又は参照されているレジスタを検出して、
前記資源割付ステップは、
未割付変数検出ステップにより検出された一時変数に、資源検出ステップにより検出されたレジスタとは異なるレジスタを割り付けることを特徴とする請求項９に記載のコンパイル方法。
【請求項１１】　前記未割付変数検出ステップは、
プログラムからアセンブラ文を検出する検出サブステップと、
アセンブラ文が検出されると、当該アセンブラ文にて定義又は参照される資源に割り付けられた割付済み一時変数を生成する割付済み一時変数生成サブステップと、
前記プログラムにおいて検出されたアセンブラ文が占めている部位を検出し、検出された占有部位を生存区間として、前記生成された割付済み一時変数に設定する生存区間設定サブステップと、
割付済み一時変数に設定された生存区間と、生存区間が重なる未割付変数を検出する未割付変数検出サブステップとからなることを特徴とする請求項９又は１０に記載のコンパイル方法。
【発明の詳細な説明】
【０００１】
【発明の属する技術分野】本発明は、ソースプログラムを機械語プログラムやアセンブラプログラムなどの目的プログラムに翻訳するコンパイラ装置に関し、特に、ソースプログラムに高級プログラミング言語で書かれたプ

ログラム部分と、アセンブラ言語で記述した部分とが含まれている場合の改良に関する。

【０００２】

【従来の技術】近年、組み込み用途向けマイクロプロセッサの高性能化により、通信、画像、音声処理など、マルチメディア処理を行なう情報機器に、組み込み用途向けマイクロプロセッサが使用されるようになりつつある。マルチメディア処理では、プログラム規模が肥大化する一方なので、プログラムの開発コスト、保守、移植性などの問題が近年に増して深刻化しており、高級プログラミング言語、特に最近ではC言語やC++言語を用いた開発環境の整備が強く要求されている。しかしマルチメディア処理の開発環境の現状を振り返ってみれば、高級プログラミング言語指向の開発環境の要求に逆行するかの如く、より機械語に近いアセンブラ言語を用いてマルチメディア処理が記述されている例が多く見られる。アセンブラ言語を用いて記述されている部分は、頻繁に実行されかつ短時間実行を強く要求されるプログラム部分であり、マルチメディア処理における大量データの処理機能の中核である。

【０００３】これは、コンパイラ自体の翻訳能力が見劣りすることも然ることながら、マイクロプロセッサに1命令で複数の機能を果たす機械語命令が存在したり、近年よく見られるマルチメディア対応命令など、高級プログラム言語記述ではどうしても効率良く記述できず、かつ、コンパイラでは効率良くそれらの命令に翻訳することが難しい場合があるためである。

【０００４】例えば、コンパイラがターゲットとするマイクロプロセッサがロードストア方式（オペランドのアドレシングにおいて、メモリ間演算命令は転送命令のみしかない方式をいう）であり、かつ、除算命令divが図１５（d）のような仕様であるものとする。図１５(d)において、除算結果は、レジスタrnに格納され、剰余算の結果は、レジスタmdrに格納されるので、多くのマイクロプロセッサの除算命令は、除算と剰余算を１命令で同時に算出することが可能である。

【０００５】これに対して高級プログラム言語であるC言語では、変数a，bの除算および剰余算の結果をそれぞれ変数c，dに求めたい場合は、図１５（a）のようにしか記述できず、その結果、多くのコンパイラは図１５（b）の(1)(2)に示すようにそれぞれの「／」演算および「%」演算に対してdiv命令を生成してしまう。div命令の実行には通常、他の命令に比べて多くの実行サイクルを要するので、理想的には、図１５（c）のように一つのdiv命令で除算、剰余算を計算することが望まれる（ここで、図１５（b）（c）では、変数a，b，c，dにそれぞれレジスタr2，r3，r4，r5が割り付けられているとしており、また、「mov rm，rn」は、レジスタrnの値をレジスタrmに転送することを意味している。）。

【０００６】そのため多くのC言語コンパイラでは、C言語表現によるプログラム記述とアセンブラ言語による記述を混在できるように、拡張言語仕様として「asm文」をプログラマに提供している。図１５の例では図１６のように「asm」キーワードに続いて、アセンブラ文による記述を可能としている。ここで、asm文内のうちC言語レベルでの変数を記述している部分は、コンパイラが割り付けたレジスタやメモリに置き換えられる。例えば、図１５と同様に変数a，b，c，dにそれぞれレジスタr2，r3，r4，r5が割り付けられた場合は図１６（b）のようにコンパイラは出力する。図１５（c）のような理想的な場合に比較して、図１６（b）では転送命令が２命令増加しているがdiv命令は１命令であり、図１５（b）よりは実行時間で有利となっている。また図１６（b）に対しては従来のコピー伝搬の最適化（後述の文献１）を適用することにより、図１５（c）のように変換される可能性もある。

【０００７】さらに、以上のように除算と剰余算を一つのdiv命令で行ないたい所が複数箇所ある場合は、図１７（a）のようにマクロ定義しておいて、同図（b）のように使用すると利便性が向上し、さらに関数呼び出しと同様の記述であるので、プログラムの読み易さも向上する。尚、図１７（b）は、図１７（a）のx，y，z，wがC言語コンパイラのマクロプロセッサによって、図１６（a）のようにa，b，c，dに置き換えられる様子を示す図である。このようなC言語コンパイラのマクロプロセッサによる置き換えをマクロ展開といい、図１７（a）のdmのように、マクロ定義されたアセンブラ文列であって、特有の機能を持つものを「インラインアセンブラルーチン」と呼ぶことにする。

【０００８】

【発明が解決しようとする課題】しかしながら、従来のコンパイラ装置において、インラインアセンブラルーチンを使用する際、プログラマには以下に示す第１の制限、第２の制限がプログラマに課せられるので、プログラマは、プログラミング時にインラインアセンブラルーチンを取り入れることを躊躇しがちである。

【０００９】ここで第１の制限では、プログラム中のある変数xの値がインラインアセンブラに跨って有効である場合に、コンパイラが変数xにインラインアセンブラで更新されるレジスタを割り付ける可能性があるため、プログラマは変数xの値が壊されていないかどうか、コンパイラが出力する目的プログラムを入念にチェックする必要がある。例えば、図１７（c）において、同図（c）の(1)で設定された変数aの値を同図（c）の(2)で使用しており、変数aの値はインラインアセンブラルーチンdmに跨って有効となっている（このように変数が保持している値が有効なプログラム区間を「変数が生きている区間」という）。このとき変数aにコンパイラがレジスタr1を割り付けたとすると、インラインアセンブラルーチンdmではレジスタr1を更新してしまうので、図１

７（ｃ）の(2)における変数aの使用時には間違った値を使ってしまうことになる。

【００１０】第２の制限では、関数呼び出しを短時間で実行するために、コンパイラの仕様として引数を特定のレジスタrで引き渡している際、その引数の値が、インラインアセンブラに跨って値が生きている場合に、インラインアセンブラでレジスタrを更新してしまう可能性があり、やはりこの場合もプログラマは、インラインアセンブラがどのように定義されているか把握した上で、引数の値が壊されることがないか入念にチェックする必要がある。例えば図１７（ｄ）のようにインラインアセンブラルーチンdmの使用箇所の後に引数pの参照があり、かつ引数pはレジスタr0で引き渡されるとすると、明らかにインラインアセンブラルーチンdmでレジスタr0の値は壊され、引数pの参照時には間違った値を使用してしまう。

【００１１】一般的に前述のようなチェックはプログラマに大きな負担を強いるものであり、その負担を軽減するために、多くのプログラマは、コンパイルする前に確実にメモリや特定のレジスタに割り付けられると判断可能な変数（大域変数など）のみが含まれている関数にインラインアセンブラルーチンを使用したり、引数渡しで使用されるレジスタをインラインアセンブラルーチン自体で定義しない、などの制限を加えて記述する場合が多い。

【００１２】しかしこのような使用上の制限は、本来の独立した機能をインラインアセンブラルーチンとして実現し、インラインアセンブラルーチンを使うことにより利便性を向上させたいというプログラマの要求に十分に答えるものでない。つまり、前述の第１、２の制限により、原則的には、インラインアセンブラルーチンがどのように定義されているかプログラマが必ずチェックする必要があるため、プログラマがインラインアセンブラルーチンの詳細な中身を知らなくても使用可能な、いわゆるブラックボックスとして使用することが困難となる。そのためインラインアセンブラルーチンをライブラリ化し、インラインアセンブラルーチンの再利用性を向上させることが困難となる。

【００１３】本発明の第１の目的は、インラインアセンブラルーチン使用時におけるチェックをプログラマに課すことのないよう、プログラムの翻訳を行うコンパイル装置を提供することである。本発明の第２の目的は、インラインアセンブラルーチンのブラックボックス化が可能なようにプログラムの翻訳を行うコンパイル装置を提供することである。

【００１４】本発明の第３の目的は、インラインアセンブラルーチンのライブラリィ化を可能とし、プログラムの再利用性を向上させることができるコンパイル装置を提供することである。

【００１５】

【課題を解決するための手段】上記第１、第２、第３の目的を達成するために、本発明は、どの資源に割り付けるべきかが未定である未割付変数のなかから、その生存区間がアセンブラ文の占有部位と重複しているものを検出する未割付変数検出手段と、未割付変数検出手段により検出された未割付変数の生存区間と重複しているアセンブラ文にて定義されている資源を検出する資源検出手段と、資源検出手段により検出された資源とは異なる資源を未割付変数検出手段により検出された未割付変数に割り付ける資源割付手段とを備えることを特徴としている。

【００１６】

【発明の実施の形態】実施形態の説明の前に実施形態に関連する文献と、用語について説明しておく。

<文献>

１．A.V.Aho, R.Sethi, J.D.Ullman:" Compilers, Principle, Techniques, andTool" , Addison Wesley Publishing Company Inc.,1986,（邦訳）原田賢一：" コンパイラI、II" ,サイエンス社,1990

２．田中：" 資源割付装置及び資源割付プログラムが記録されたコンピュータ読み取り可能な記録媒体" ，出願番号 特願平１０－３４４５２４

３．Andrew W.Appel:" Modern Compiler Implementation in Java" , CAMBRIDGEUNIVERSITY PRESS, 1998

<用語の説明>

・変数の定義、参照、使用

変数の値を設定することを「定義する」と呼び、設定した値を使用することを「参照する」と呼ぶ。またプログラム上で変数を定義したり、参照したりすることを変数を「使用する」と呼ぶことにする。また、以下では仮引数も特に断らない限り変数と同様に扱う。

【００１７】・一時変数

処理をし易すくするためにコンパイラが計算結果を一時的に格納するために生成する変数のことを「一時変数」と呼ぶことにする。以下では特に断らない限り通常のソースプログラムで記述される変数と同様に扱う。

・中間命令

処理をし易すくするためにコンパイラがソースプログラムのコードを中間的なコードに変換したものを「中間コード」と呼び、中間コードの１ステップを「中間命令」と呼ぶことにする。中間命令には４つ組や３つ組などが存在し、これらが変換されて最終的な目的プログラムが生成される。またソースプログラムを中間命令の列に変換したものを「中間プログラム」と呼ぶことにする。図７は図６のC言語プログラムに対する中間プログラムの一例であり、４つ組の一つである３番地コードで表現している。また図７のi1，i2など、「i」とそれに付随する番号で示したものは、中間命令の識別子を意味している。 特に、図６の「a =p+b+10」 は図７では中間命令i1，i2というように一時変数t1を介して分割さ

れ、３番地コードとして表現されている。また、図６(1)のasm文は、図７(1)のように図面上は図６(1)と同一な形式の中間命令に変換されている。特に図７(1)のasm文の中間命令を「asm文中間命令」と呼ぶことにする。尚、本実施形態における中間命令は特にこの形式に限られるわけでなく他の形式の中間命令にも適用可能である。また、コンパイラ装置内の処理を簡単化するために、中間プログラムの最初と終りを表す中間命令や、関数の始めと終りを表す中間命令など特定のプログラム範囲を表す中間命令も用意されている。図７では中間命令i５０，i５１のように表現される関数の関数の始めと終りを示す中間命令が示されている。

【００１８】・基本ブロック

中間命令列のうち、中間命令列内部からの飛び越しも、また中間命令列内部への飛び越しもなく、中間命令列の実行順が一定順序である場合、その中間命令列を基本ブロックという。基本ブロックについては文献１に詳細に記載されている。

【００１９】・生存区間

生存区間とはそれぞれの変数に保持されている値が有効となる区間であり、正確には変数に値を定義する中間命令からその定義された値を最後に参照する中間命令までのプログラム上の区間のことである。そして、生存区間はこの区間に含まれる中間命令の集合で表される。ここで変数に値を定義する中間命令は生存区間の開始点に相当し、生存区間中で定義された値を参照している中間命令のうち、最後に参照する中間命令は生存区間の終了点に相当する。また生存区間の重複、非重複を中間命令の集合積で判定するため、二つの生存区間の終始が一致する中間命令では、それら二つの生存区間は重ならないとする。図９（b）が生存区間の一例である。図９（b）において生存区間は縦線の実線で表されている。生存区間の開始点、及び終了点をそれぞれ白丸、黒丸で表現している。また図１８は図９（b）の生存区間を表現する具体的なデータ構造を示しており、各変数の生存区間が中間命令の集合として示されている。図１８の「生存区間」における「..」は、図外に何らかの中間命令が存在することを示す。生存区間の詳細な定義および具体例は文献１，２に記載されている。

【００２０】・変数への資源割り付け

変数への資源割り付けとは、変数にレジスタやメモリを割り付けることである。一つのレジスタやメモリにはプログラムの同一区間で二つ以上の値を保持することは不可能であるため、生存区間が重なる変数には同じレジスタやメモリを割り付けることは不可能であり、異なるレジスタやメモリを割り付けることが必須となる。例えば、図９の変数aと変数bの二つの生存区間が重なっており、それぞれの変数には異なるレジスタやメモリを割り付ける必要がある。以下ではレジスタやメモリを合わせて「資源」と呼ぶことにする。

【００２１】<第１実施形態>図１は第１実施形態のコンパイラ装置の構成図である。コンパイラ装置１は、マクロ展開部１１、構文解析部１２、最適化部１３、資源割付部１４、コード生成部１５の一般的なコンパイラ装置を構成する主要な部分と、レジスタ引数置換部１６、asm文定義レジスタ検出部１７、asm文定義レジスタ保持部１８からなり、これらは何れも本実施形態に関係している部分で構成されている。

【００２２】次に各部の動作と機能について例を示しながら説明する。マクロ展開部１１は、コンパイラ装置１により、まず最初に起動されるものであり、マクロ定義されたマクロ識別子をその本体で置き換える。この処理部分は一般的なC言語コンパイラのマクロプロセッサの処理と同様なので詳細説明は省略し処理結果の例について説明する。

【００２３】図５はソースプログラムの一例であり、「従来の技術」で取り上げた除算と剰余算を行なうインラインアセンブラルーチンをマクロ定義したdm（x，y，z，w）と、そのdmを使用している関数fの例を示している。尚関数fでは本実施形態の説明に使用する部分のみを抽出して記述している。図６は図５のソースプログラムをマクロ展開部１１で処理した後の中間的なプログラムを示している。図６(1)に示すようにマクロ識別子dmがマクロ定義されたインラインアセンブラルーチン本体に置き換えられている。

【００２４】図６の(1)は、「mov a,r0」「mov b,r1」「div r1,r0」「mov r0,c」「mov mdr,d」という5つのアセンブラ命令を含むが、これらは何れも文頭に「asm」キーワードが付与されていることがわかる。この「asm」キーワードは、当該「asm」キーワードから文末のセミコロン記号までに示されている一文が、アセンブラ命令を含んでおり（このように、「asm」キーワードが付与され、アセンブラ命令を含んでいる文をアセンブラ文と呼ぶ。）、この文は、通常の高級言語記述されたプログラム部分と同等な翻訳処理が不要な旨をコンパイラ装置に指示している。そのためコンパイラ装置における構文解析部１２及び最適化部１３は、文頭に「asm」キーワードが付与された文についての処理をスキップする。

【００２５】構文解析部１２は、マクロ展開部１１の処理終了後に起動され、マクロ展開部１１によりマクロ展開されたプログラムの字句解析、構文解析および意味解析を行い、中間プログラムに変換する。例えば、図６のプログラムは図７の中間プログラムに変換される。この構文解析部１２の詳細は本実施形態の主眼ではないため説明を省略する。この際、マクロ展開部１１により、「asm」キーワードが付与されてソースプログラム内に展開されたアセンブラ文は、構文解析部１２によって他のコードに置き換えられることもなく、そのまま変換後の中間プログラム内に継承される。これにより、「asm」キーワードが付与されたアセンブラ文は、中間命令として、

中間プログラム内に存在することになる。

【００２６】最適化部１３は、構文解析部１２の処理終了後に起動され、最終的にコンパイラ装置１が生成する目的プログラムのプログラムサイズ縮小化及び処理実行時間の短縮化を図るよう中間プログラムの最適化を行う。この最適化部１３の詳細は本実施形態の主眼ではないため説明を省略し、本実施形態に関連のある点についてのみ説明する。最適化部１３では、最適化を行なうために、基本ブロック解析、制御フロー解析、データフロー解析という処理を行なう。基本ブロック解析とは処理対象の中間プログラムを基本ブロックに分割することである。制御フロー解析とは各基本ブロック間の制御の流れを解析することである。データフロー解析とは各基本ブロック中、それぞれの変数がどこで定義され、どこで参照されているかについて解析することである。またこれらの解析結果から、後述する資源割付部１４は生存区間を算出する。尚、翻訳すべきプログラムにレジスタ引数が含まれている場合、最適化部１３は、レジスタ引数置換部１６を起動した後に起動される。この際、「asm」キーワードが付与されて中間プログラム内に継承されたアセンブラ文は、最適化部１３の最適化により他のコードに置き換えられることなく、また、削除されることもなく、そのまま最適化後の中間プログラム内に継承される。

【００２７】レジスタ引数置換部１６は、翻訳すべきプログラムにレジスタ引数が含まれている場合に起動され、レジスタで渡される仮引数の値を一旦一時変数に格納する中間命令を関数の最初に挿入し、さらに中間プログラムの仮引数の使用部分を全て一時変数に置き換える。図２はレジスタ引数置換部１６の処理フローチャートである。

【００２８】ステップa1では、全ての仮引数pについてステップa2からステップa5を繰り返し、全てについて処理を終えたらレジスタ引数置換部１６の処理を終了する。ステップa2では、仮引数pがレジスタで渡されるものであるとき、ステップa3からステップa4を行なう。ステップa3では、新たに一時変数tを生成し、仮引数pから一時変数tへの代入を示す中間命令i：t=p；を生成する。

【００２９】ステップa4では生成した中間命令iを関数の最初に挿入する。ステップa5では、中間プログラムの仮引数pの全ての使用部分を一時変数tで置き換え、ステップa1に戻る。次にレジスタ引数置換部１６の具体的な動作例を図８の例を用いて説明する。図８は、図７の中間プログラムに対してレジスタ引数置換部１６の処理を行なった結果である。まずレジスタで引き渡される仮引数pが取り出され（ステップa1、a2）、仮引数pから新たに生成された一時変数t2への代入を示す中間命令i２０が生成され（ステップa3）、中間命令i２０が中間プログラムの最初に挿入され（ステップa4）、中間命令i1，i9において仮引数pが使用されている部分が一時変数t2に置き換えられている（ステップa5）。

【００３０】資源割付部１４は、最適化部１３の処理終了後に起動され、生存区間が重なる複数の変数それぞれに、互いに異なるレジスタを割り付ける。ここで、プログラム内にアセンブラ文が展開されている場合、資源割付部１４は自身が処理を行う前にasm文定義レジスタ検出部１７を起動する。asm文定義レジスタ検出部１７は、中間プログラムの何れかの中間命令にasm識別子が付与されたアセンブラ文が含まれている場合、当該アセンブラ文で定義されるレジスタを検出して、検出結果をasm文定義レジスタ保持部１８に格納する。図３はasm文定義レジスタ検出部１７の処理フローチャートである。

【００３１】ステップb1では、全ての中間命令iについてステップb2からステップb3を繰り返し、全ての中間命令iについて処理を終えたらasm文定義レジスタ検出部１７の処理を終了する。ステップb2では、中間命令iがasm文中間命令であるときステップb3を行なう。そうでないときはステップb1へ戻る。

【００３２】ステップb3では、中間命令iでレジスタrが定義されている場合、中間命令iとレジスタrをasm文定義レジスタ保持部１８に格納し、ステップb1へ戻る。図１０はasm文定義レジスタ保持部１８の図８の中間プログラムの例に対する保持内容である。asm文中間命令毎に定義するレジスタを保持しており、例えば、図８のasm文中間命令i３では変数aの値がレジスタr0に転送され、レジスタr0が定義されるので、図１０のasm文中間命令i３の定義されるレジスタにはr0が設定される（ステップb3）。

【００３３】資源割付部１４は、最適化部１３で解析された制御フロー情報とデータフロー情報を用いて、前述のレジスタ引数置換部１６で生成した一時変数も含めて、全ての変数の生存区間を算出する。生存区間の算出方法は文献１、２などと同様なので詳細説明は省略し、ここでは、図８の中間プログラム例に対する生存区間の算出結果を示すにとどめる。図９が図８の中間プログラムに対して生存区間を算出した結果である。生存区間は、文献２などように一般的には中間命令の集合として表現されるが、ここでは直観的に解り易いように実線でその範囲を示している。また、図９（b）の仮引数pの生存区間に記載されている（r0）は仮引数pにレジスタr0が割り付けられていることを意味している。生存区間を算出した後、資源割付部１４は、前述のレジスタ引数置換部１６で生成した一時変数も含めた変数への資源割り付けを行なう。特に資源割付部１４は、変数へのレジスタ割り付けにおいて、生存区間が重なる変数に割り付けられているレジスタでなくかつ、変数の生存区間にasm文中間命令が含まれているとき、asm文定義レジスタ保持部１８から得られるasm文中間命令が定義するレジスタでもないレジスタを割り付ける。図４が資源割付部１

４の処理フローチャートである。

【００３４】ステップｃ１では、全ての資源が未割付けの変数ｖについてステップｃ２からステップｃ６まで繰り返し、処理を終えたら資源割付部１４の処理を終了する。ステップｃ２では、変数ｖと生存区間が重なる変数に割り付けられているレジスタの集合Ｒ１を求める。ステップｃ３では、変数ｖの生存区間に含まれる全てのasm文中間命令ｉについて、asm文中間命令ｉで定義されるレジスタをasm文定義レジスタ保持部１８から検出し、レジスタの集合Ｒ２に求める。

【００３５】ステップｃ４では、ステップｃ２で求めたレジスタ集合Ｒ１にも、ステップｃ３で求めたレジスタ集合Ｒ２にも両方に属さないレジスタｒが存在するときはステップｃ５を実行し、そうでないときはステップｃ６を実行する。ステップｃ５では、変数ｖにレジスタｒを割り付け、ステップｃ１に戻る。ステップｃ６では、変数ｖにメモリを割り付け、ステップｃ１に戻る。

【００３６】次に資源割付部１４の具体的な動作例を図９の例を用いて説明する。ここでは変数ａと変数ｔ２の割り付けについて述べる。まず変数ａと生存区間が重なる変数ｔ２，ｂ，ｃ，ｄに割り付けられているレジスタの集合Ｒ１を求めるが、ここでは、変数ｔ２，ｂ，ｃ，ｄには未だ何も割り付けられていないとするので、集合Ｒ１は空集合となる（ステップｃ２）。次に変数ａの生存区間に含まれているasm文中間命令ｉ３からｉ７について、図１０のasm文定義レジスタ保持部１８を参照して、それぞれのasm文中間命令で定義されるレジスタｒ０，ｒ１，ｍｄｒを集合Ｒ２に取り出す（ステップｃ２）。求めたレジスタの集合Ｒ１、Ｒ２に属さないレジスタ、例えばレジスタｒ２を変数ａに割り付ける（ステップｃ４、ｃ５）。

【００３７】次に引き続き変数ｔ２の割り付けについて説明する。まず変数ｔ２と生存区間が重なる変数ｔ１，ａ，ｂ，ｃ，ｄ，ｅに割り付けられているレジスタの集合Ｒ１を求める。ここでは、変数ｔ１，ｂ，ｃ，ｄ，ｅには未だ何も割り付けられていないとし、変数ａが先に述べた通りレジスタｒ２に割り付けられているとすると、集合Ｒ１にはレジスタｒ２が取り出される（ステップｃ２）。次に変数ｔ２の生存区間に含まれているasm文中間命令ｉ３からｉ７について、図１０のasm文定義レジスタ保持部１８を参照して、それぞれのasm文中間命令で定義されるレジスタｒ０，ｒ１，ｍｄｒを集合Ｒ２に取り出す（ステップｃ２）。求めたレジスタの集合Ｒ１、Ｒ２に属さないレジスタ、例えばレジスタｒ３を変数ｔ２に割り付ける（ステップｃ４、ｃ５）。

【００３８】次にコンパイラ装置１はコード生成部１５を起動し、中間命令をコンパイラがターゲットとするマシンのアセンブラ文又は機械語命令などの目的プログラムに変換する。コード生成部１５自体は従来方式と同様なので詳細説明は省略する。以上述べたように本実施形態によれば、図９の変数ａ，ｔ２のようにasm文を跨って生きている変数には、資源割付部１４のステップｃ３からｃ５において、asm文で定義されるレジスタは割り付けられないので、前述の「発明が解決しようとする課題」で述べた第１の制限は無くなり、またレジスタ引数置換部１６のステップａ２からａ５により、図９の一時変数ｔ２が生成され、さらに一時変数ｔ２には資源割付部１４のステップｃ３からｃ５において、asm文で定義されるレジスタは割り付けられないので、「発明が解決しようとする課題」で述べた第２の制限も無くなる。よって、プログラマが自由にインラインアセンブラルーチンを定義し、それをブラックボックスとして任意の場所で使用可能となり、インラインアセンブラルーチン使用時のプログラマのチェックの負担を大幅に軽減し、かつ、インラインアセンブラルーチンをライブラリ化し、プログラムの再利用性を向上させることが可能となる。

【００３９】尚、レジスタ引数置換部１６のフローチャートのステップａ４では、生成した中間命令ｉを関数の最初に挿入するようにしているが、仮引数ｐの値が参照される前の位置に挿入してもよい。

<第２実施形態>図１１は第２実施形態のコンパイラ装置の構成図である。コンパイラ装置２は、マクロ展開部２１、構文解析部２２、最適化部２３、資源割付部２４、コード生成部２５といった一般的なコンパイラ装置を構成する主要な部分と、レジスタ引数置換部２６、asm文定義レジスタ検出部２７、asm文定義レジスタ保持部２８、asm文生存区間生成部２９といった本実施形態に関係している部分で構成されている。

【００４０】コンパイラ装置２は、第１実施形態で述べたコンパイラ装置１と同様に、マクロ展開部２１、構文解析部２２、最適化部２３と順番に起動する。ここで、図２のマクロ展開部２１、構文解析部２２、最適化部２３および、レジスタ引数置換部２６は、第１実施形態で説明した図１のマクロ展開部１１、構文解析部１２、最適化部１３およびレジスタ引数置換部１６と同じものであるので、以下では資源割付部２４、asm文定義レジスタ検出部２７、asm文生存区間生成部２９に焦点を絞った説明を行なう。よって、図５のプログラム例に関しては、第１実施形態と同様に、最適化部２３の処理を終了した時点では、図８のような中間プログラムが生成されているものとする。

【００４１】asm文定義レジスタ検出部２７は、第１実施形態で説明したasm文定義レジスタ検出部１７と同一構成であり、まず最初に資源割付部２４により起動される。図８の中間プログラム例に対して、asm文定義レジスタ検出部２７が検出処理を行うと、図１０のようにasm文定義レジスタ保持部２８の保持内容が設定される。資源割付部２４は、asm文定義レジスタ検出部２７が検出処理を行った後、最適化部２３で解析された制御フロー情報とデータフロー情報を用いて全ての変数の生存区間を算出する。図８の例に対する処理結果は第１実施形

態で説明した図９と同様である。

【００４２】asm文生存区間生成部２９は、資源割付部２４が生存区間を算出した後に起動され、asm文で定義されるレジスタに割り付けられた割付済みの一時変数とその生存区間を生成する。図１２がasm文生存区間生成部２９の処理フローチャートである。ステップd1では、全てのasm文中間命令iについてステップd2からステップd5まで繰り返し、処理を終えたらasm文生存区間生成部２９の処理を終了する。

【００４３】ステップd2では、asm文定義レジスタ保持部２８からasm文中間命令iで定義されるレジスタrが存在するときステップd3からステップd5を行ない、そうでないときはステップd1へ戻る。ステップd3では、新たに一時変数sを生成する。ステップd4では、生成した一時変数sに対してasm文中間命令iで生存区間が開始し終了する生存区間を生成する。

【００４４】ステップd5では、生成した一時変数sにレジスタrを割り付け、ステップd1へ戻る。図１４（b）は、asm文生存区間生成部２９の処理後の各変数の生存区間を示しており、図９（b）に対してさらにレジスタに割り付けられている一時変数s1，s2，s3とその生存区間が付け加わっている。例えばasm文中間命令i３においては、asm文中間命令i３で定義されるレジスタr0を割り付けた一時変数s1と、一時変数s1に対してasm文中間命令i３で生存区間が開始し終了している生存区間が生成されている。

【００４５】次に資源割付部２４は変数への資源割り付けを行なう。図１３が資源割付部２４の処理フローチャートである。ステップe1では、全ての資源が未割付けの変数vについてステップe2からステップe5まで繰り返し、全ての変数vについて処理を終えたら資源割付部２４の処理を終了する。

【００４６】ステップe2では、変数vと生存区間が重なる変数に割り付けられているレジスタの集合Rを求める。ステップe3では、ステップe2で求めたレジスタの集合Rに属さないレジスタrが存在するときはステップe４を実行し、そうでないときはステップe5を実行する。

【００４７】ステップe4では、変数vにレジスタrを割り付け、ステップe1に戻る。ステップe5では、変数vにメモリを割り付け、ステップe1に戻る。次に資源割付部２４の具体的な動作例を図１４の例を用いて説明する。ここでは変数aと変数t2の割り付けについて述べる。変数aと生存区間が重なる変数はt2，b，c，d，s1，s2，s3であるが、このうち変数s1，s2，s3には、それぞれレジスタr0，r1，r0とmdrが割り付けられているので、レジスタの集合Rにはこれらレジスタが属することになり（ステップe2）、変数aには、それ以外の例えば、レジスタr2が割り付けられる。

【００４８】次に引続き変数t2の割り付けについて説明する。変数t2に関しても同様に、変数t2と生存区間が重なる変数は、変数t1，a，b，c，d，e，s1，s2，s3であり、このうちレジスタに割り付けられているのは、変数a，s1，s2，s3でありそれぞれレジスタr2，r0，r1，r0とmdrが割り付けられているので変数t2には、それ以外の例えばレジスタr3が割り付けられる。

【００４９】次にコンパイラ装置２はコード生成部２５を起動し、中間命令をコンパイラがターゲットとするマシンのアセンブラ文又は機械語命令などの目的プログラムに変換する。コード生成部２５自体は従来方式と同様なので詳細説明は省略する。以上述べたように本実施形態によれば、図１４の変数a，t2のようにasm文を跨って生きている変数には、asm文生存区間生成部２９で生成される一時変数s1，s2，s3の効果により、資源割付部２４のステップe3，e4において、asm文で定義されるレジスタは割り付けられないので前述の「発明が解決しようとする課題」で述べた第１の制限は無くなり、またレジスタ引数置換部１６のステップa2からa5により、図１４の一時変数t2が生成され、さらに一時変数t2には資源割付部２４のステップe3、e4において、asm文で定義されるレジスタは割り付けられないので、「発明が解決しようとする課題」で述べた第２の制限も無くなる。よって、プログラマが自由にインラインアセンブラルーチンを定義し、それをブラックボックスとして任意の場所で使用可能となり、インラインアセンブラルーチン使用時のプログラマのチェックの負担を大幅に軽減し、かつ、インラインアセンブラルーチンをライブラリ化し、プログラムの再利用性を向上させることが可能となる。

【００５０】尚、前述のasm文生存区間生成部２９によって生成される一時変数sの導入とその生存区間の生成によって、資源割付部２４に従来方式の文献２、３と同様の資源割り付け方式が適用可能となる。文献２に関しては、図１４の中間プログラムと生存区間の情報をそのまま用いることによって適用可能となる。特に文献２は、コンパイラが生成する目的プログラの転送命令の発生を極力押えることが可能であり、例えば、図１４の変数bと変数s2のように生存区間の終始が一致している変数は、可能であれば同じレジスタに割り付けられるので、変数bがレジスタr1に割り付けられ、asm中間命令i４は同一レジスタ間の転送となり削除される。

【００５１】また、広く知られている「グラフカラーリング」方式の拡張方式である文献３に関しては、図１４の生存区間の情報から図１９のような変数間の生存区間の重なりを示すグラフである干渉グラフを構築することによって適用可能となる。図１９のグラフは、変数をグラフのノードとして表現し、生存区間が重なる変数（ノード）をエッジで結んだものであり、特にグラフのノードのうち一部のノード、図１９ではノード（変数）p，s1，s2，s3が既にカラーリング（レジスタ割り付け）されている。文献３の方式は特に、このように既に一部の

ノードがカラーリングされているグラフを扱うことの可能な「グラフカラーリング方式」である。

【００５２】また以上述べた実施形態では除算命令に関するものついて述べたが、単に除算命令に対してのみ適用できるのでなく、積和演算命令などのマルチメディア処理対応の高機能命令や、四則演算、ビット操作演算等の任意の命令であって良く、また、それら複数の命令の組み合わせであっても良いことは言うまでもない。更に、本実施形態でフローチャートを参照して説明した手順（図２～図４、図１２～図１３）等を機械語プログラムにより実現し、これを記録媒体に記録して流通･販売の対象にしても良い。このような記録媒体には、ICカードや光ディスク、フロッピー（登録商標）ディスク等があるが、これらに記録された機械語プログラムは汎用コンピュータにインストールされることにより利用に供される。この汎用コンピュータは、インストールした機械語プログラムを逐次実行して、本実施形態に示したコンパイラ装置の機能を実現するのである。

【００５３】

【発明の効果】　以上のように本発明は、どの資源に割り付けるべきかが未定である未割付変数のなかから、その生存区間がアセンブラ文の占有部位と重複しているものを検出する未割付変数検出手段と、未割付変数検出手段により検出された未割付変数の生存区間と重複しているアセンブラ文にて定義されている資源を検出する資源検出手段と、資源検出手段により検出された資源とは異なる資源を未割付変数検出手段により検出された未割付変数に割り付ける資源割付手段とを備えているので、インラインアセンブラルーチン使用時におけるチェックがプログラマに課せられることはない。インラインアセンブラルーチン使用時において従来プログラマに課せられていたチェック負担がなくなるので、アセンブラ文をプログラム上の任意の位置に記述することが可能となり、インラインアセンブラルーチンもプログラム上の任意の位置に記述するできる。それに伴い、インラインアセンブラルーチンを積極的に使用することによるプログラムの高速実行性を向上させることができ、プログラムの再利用性を向上させることができる。

【００５４】ここで何れかの関数にレジスタを用いて引渡される仮引数が存在する場合、当該仮引数の値を一時変数に代入する旨の代入命令を生成して当該関数の最初の位置に挿入すると共に、関数において使用されている全ての仮引数を、前記代入命令において値が代入される一時変数に置き換えるレジスタ引数置換手段を備え、前記未割付変数検出手段は更に、解析された生存区間内にアセンブラ文が含まれている一時変数を検出し、前記資源検出手段は更に、生存区間に含まれている当該アセンブラ文にて定義又は参照されているレジスタを検出して、前記資源割付手段は、未割付変数検出手段により検出された一時変数に、資源検出手段により検出されたレジスタとは異なるレジスタを割り付けてもよい。この場合、レジスタで渡される仮引数に割り付けられているレジスタが、アセンブラ文で不正に更新されることはなく、また、アセンブラ文を跨って生きている変数もアセンブラ文で不正に更新されることはなくなる。これによりインラインアセンブラルーチンを自由に定義し、それをブラックボックスとして任意の場所で使用可能となり、インラインアセンブラルーチンを積極的に使用することによるプログラムの高速実行性を向上させるとともにプログラムの再利用性も向上させることが可能となる。

【００５５】また上記装置において、前記未割付変数検出手段は、前記プログラムからアセンブラ文を検出する検出部と、アセンブラ文が検出されると、当該アセンブラ文にて定義又は参照される資源に割り付けられた割付済み一時変数を生成する割付済み一時変数生成部と、前記プログラムにおいて検出されたアセンブラ文が占めている部位を検出し、検出された占有部位を生存区間として、前記生成された割付済み一時変数に設定する生存区間設定部と、割付済み一時変数に設定された生存区間と、生存区間が重なる未割付変数を検出する未割付変数検出部とを備えているので、アセンブラ文から割付済み一時変数を生成して、これを取り込む形式で資源割り付けを行うことができる。故に、転送命令の発生を極力押えることができる文献２と同様の資源割り付け方式や、広く知られている「グラフカラーリング」方式の拡張方式である文献３と同様の資源割り付け方式を適用することができる。

【図面の簡単な説明】

【図１】第１実施形態に係るコンパイラ装置１の構成図である。

【図２】レジスタ引数置換部１６の処理のフローチャートである。

【図３】asm文定義レジスタ検出部１７の処理手順を示すフローチャートである。

【図４】資源割付部１４の処理手順を示すフローチャートである。

【図５】高級言語で書かれたプログラムの一例を示す図である。

【図６】マクロ展開部１１を実行した後のプログラムを示す図である。

【図７】構文解析部１２を実行した後の中間プログラムの例を示す図である。

【図８】レジスタ引数置換部１６を実行した後の中間プログラムを示す図である。

【図９】プログラム例に対する中間プログラムと生存区間を示す図である。

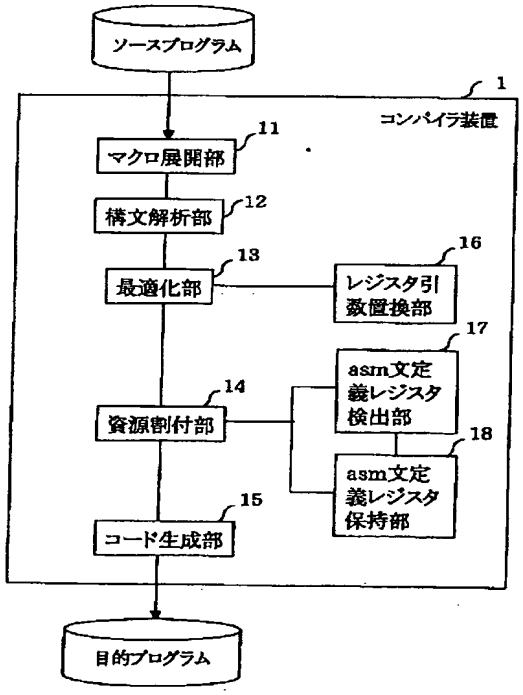
【図１０】asm文定義レジスタ保持部１８および２８の保持内容の一例を示す図である。

【図１１】第２の実施形態に係るコンパイラ装置２の構

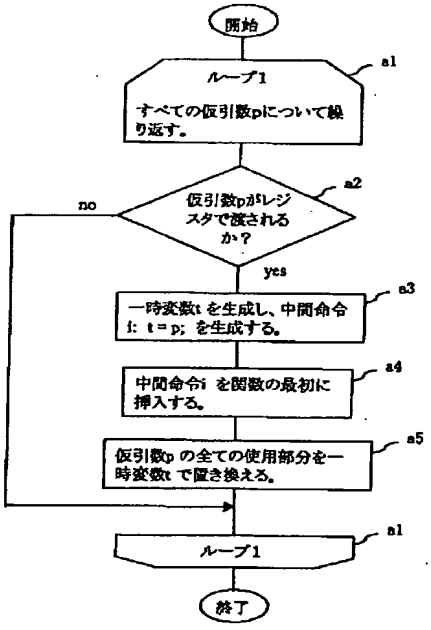
成図である。
【図１２】asm文生存区間生成部２９の処理手順を示すフローチャートである。
【図１３】資源割付部２４の処理手順を示すフローチャートである。
【図１４】asm文生存区間生成部２９を実行した後のプログラム例に対する中間プログラムと生存区間を示す図である。
【図１５】プログラム例と、対応するアセンブラプログラムと、比較アセンブラプログラムを示す図と、除算命令を説明する図である。
【図１６】asm文の記述例と、対応するアセンブラプログラムを示す図である。
【図１７】インラインアセンブラルーチンの定義例と、インラインアセンブラルーチンの使用例を示す図である。
【図１８】生存区間を表現する具体的なデータ構造を示す図である。
【図１９】図１４の生存区間に対する変数の干渉グラフを示す図である。

＊【符号の説明】
１　コンパイラ
１１　マクロ展開部
１２　構文解析部
１３　最適化部
１４　資源割付部
１５　コード生成部
１６　レジスタ引数置換部
１７　asm文定義レジスタ検出部
１８　asm文定義レジスタ保持部
２　コンパイラ
２１　マクロ展開部
２２　構文解析部
２３　最適化部
２４　資源割付部
２５　コード生成部
２６　レジスタ引数置換部
２７　asm文定義レジスタ検出部
２８　asm文定義レジスタ保持部
２９　asm文生存区間生成部

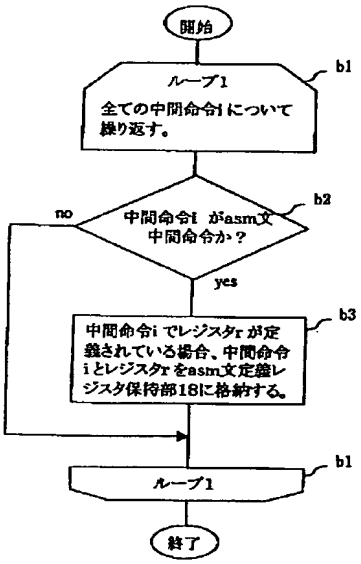
【図１】



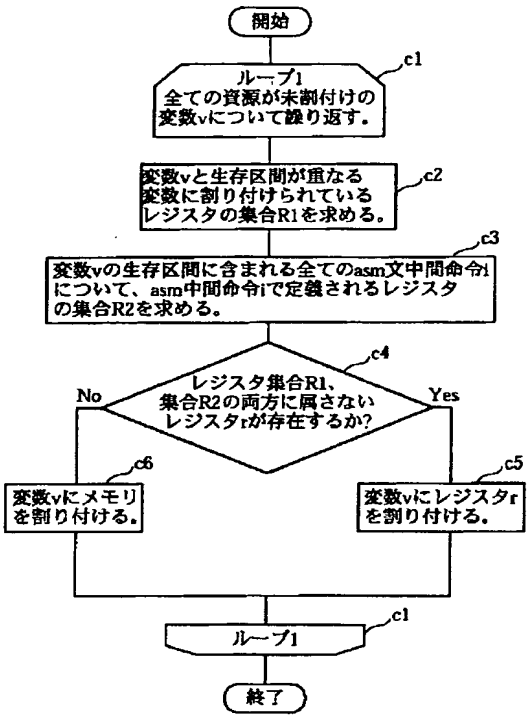
【図２】



【図３】



【図４】



【図１６】

(a)

```
int f (int p){
   int a,b,c,d;
   .....
    asm  mov a,r0  ;
    asm  mov b,r1  ;
    asm  div r1,r0 ;
    asm  mov r0,c  ;
    asm  mov mdr,d ;
   .....
}
```

(b)

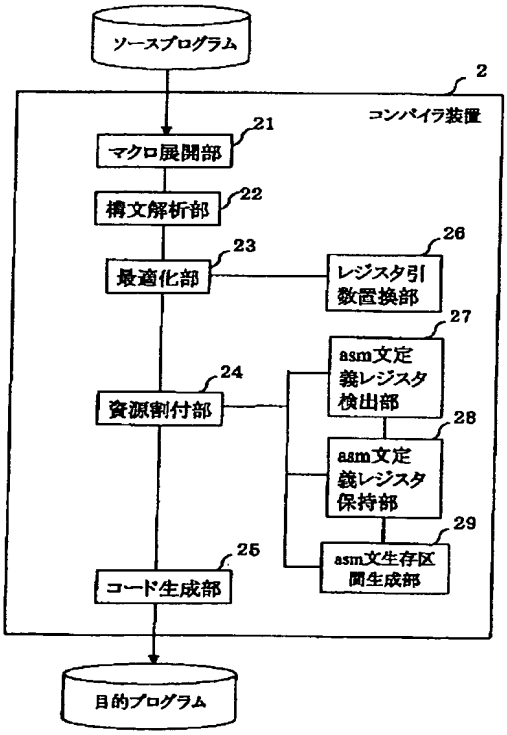
```
   .....
    mov r2,r0
    mov r3,r1
    div r1,r0
    mov r0,r4
    mov mdr,r5
   .....
```

【図１０】

asm文中間命令	定義レジスタ
...	...
i3	r0
i4	r1
i5	r0,mdr
i6	-
i7	-
...	...

asm文定義レジスタ保持部18および28の保持例

【図１１】



【図５】

```
#define     dm(x,y,z,w) ¥
            asm         mov   x, r0 ;¥
            asm         mov   y, r1 ;¥
            asm         div   r1,r0 ;¥
            asm         mov   r0,z  ;¥
            asm         mov   mdr,w
   ..........

int f (int p){
    int a,b,c,d;
   ..........
    a = p + b + 10;
   ..........
    dm(a,b,c,d);      (1)
   ..........
    e =  a + 20;
   ..........
    g =  p + 30;
   ..........
    h =  c;
   ..........
    i = d + h;
   ..........
}
```

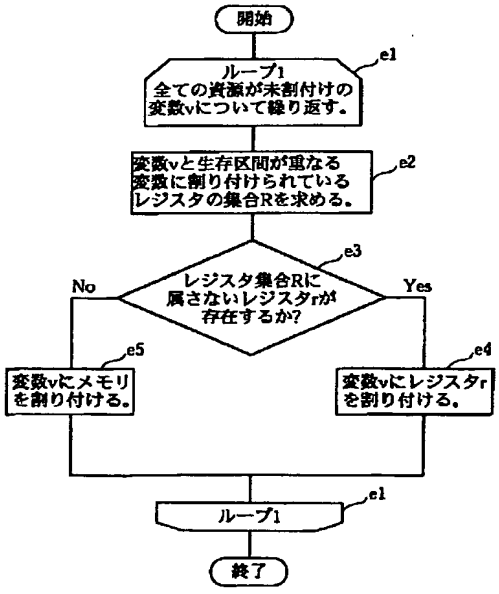
【図７】

```
        ..........
i50: Function f
        ..........
i1:  t1 = p  + b;
i2:  a  = t1 + 10;
        ..........
i3:  asm  mov a,r0  ;
i4:  asm  mov b,r1  ;
i5:  asm  div r1,r0 ;
i6:  asm  mov r0,c  ;
i7:  asm  mov mdr,d ;
        ..........
i8:  e =  a + 20;
        ..........
i9:  g =  p + 30;
        ..........
i10: h =  c;
        ..........
i11: i =  d + h;
        ..........
i51: Function End
        ..........
```

マクロ定義の本体

(1)

【図１３】



中間プログラム

【図１８】

変数	生存区間
t1	i1,i2
t2	i1,i2,..,i3,i4,i5,i6,i7,..i8,..i9
a	i2,..i3,i4,i5,i6,i7,..i8
b	..,i1,i2,..i3,i4
c	i6,i7,..i8,..i9,i10
d	i7,..i8,..i9,..i10,..i11
e	i8,..i9,..i10,..i11,..
g	i9,..i10,..i11,..
h	i10,..i11
i	i11..
p	...i20

【図６】

```
.....
int f (int p){
    int a,b,c,d;
    .....
    a = p + b + 10;
    .....
    asm  mov a,r0  ;
    asm  mov b,r1  ;
    asm  div r1,r0 ;      (1)
    asm  mov r0,c  ;
    asm  mov mdr,d ;
    .....
    e =  a + 20;
    .....
    g =  p + 30;
    .....
    h =  c;
    .....
    i =  d + h;
    .....
}
.....
```

【図８】

```
     .....
i50: Function  f
i20: t2 = p;
     .....
i1:  t1 = t2  + b;
i2:  a  = t1 + 10;
     .....
i3:  asm  mov a,r0  ;
i4:  asm  mov b,r1  ;
i5:  asm  div r1,r0 ;      (1)
i6:  asm  mov r0,c  ;
i7:  asm  mov mdr,d ;
     .....
i8:  e =  a + 20;
     .....
i9:  g =  t2 + 30;
     .....
i10: h =  c;
     .....
i11: i =  d + h;
     .....
i51: Function End
     .....
```

中間プログラム

【図１９】

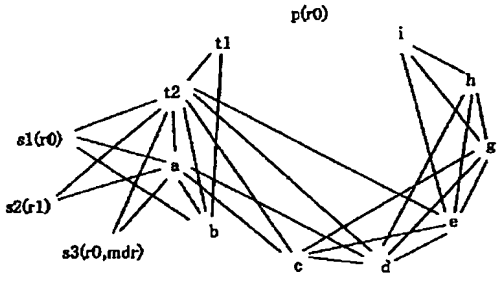
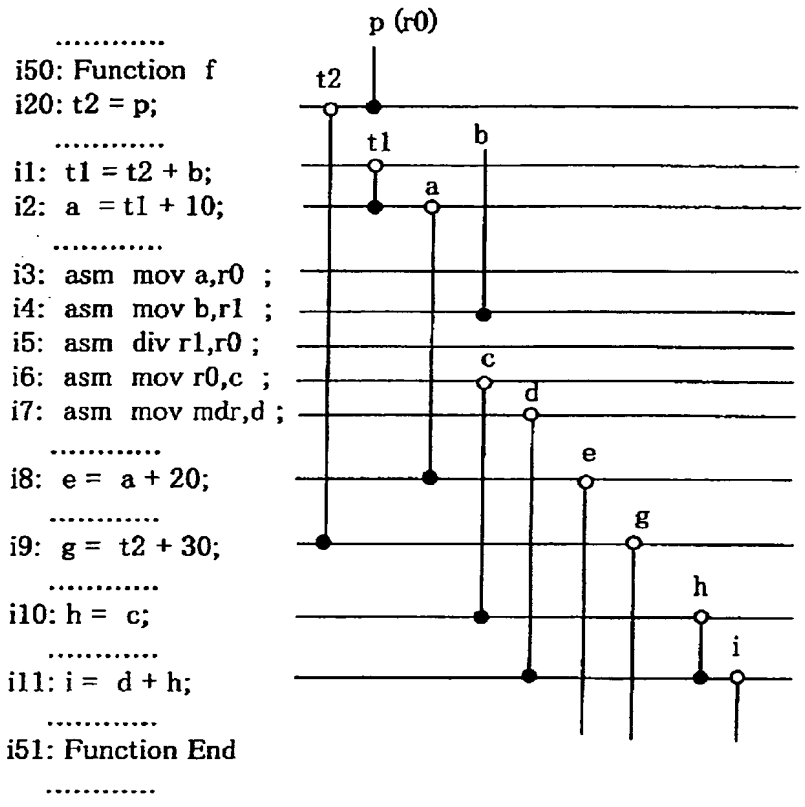


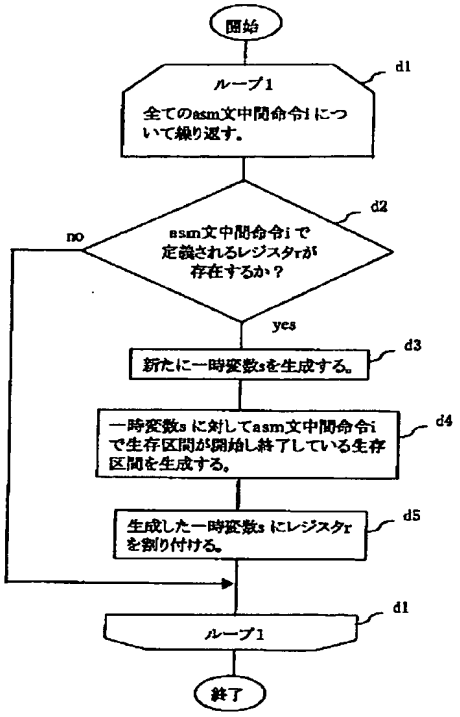
図13に対応する干渉グラフ

【図９】



(a)中間プログラム　　(b)生存区間

【図１２】



【図１７】

(a)

```
#define    dm(x,y,z,w) ¥
           asm      mov   x, r0 ;¥
           asm      mov   y, r1 ;¥
           asm      div   r1,r0 ;¥
           asm      mov   r0,z  ;¥
           asm      mov   mdr,w
```

(b)

```
int f (int p){
    int a,b,c,d;
    ..........
    dm(a,b,c,d);
    ..........
}
```

(c)

```
int f (int p){
    int a,b,c,d,e;
    ..........
    a = p + b;      (1)
    ..........
    dm(a,b,c,d);
    ..........
    e = a - 10;     (2)
    ..........
}
```

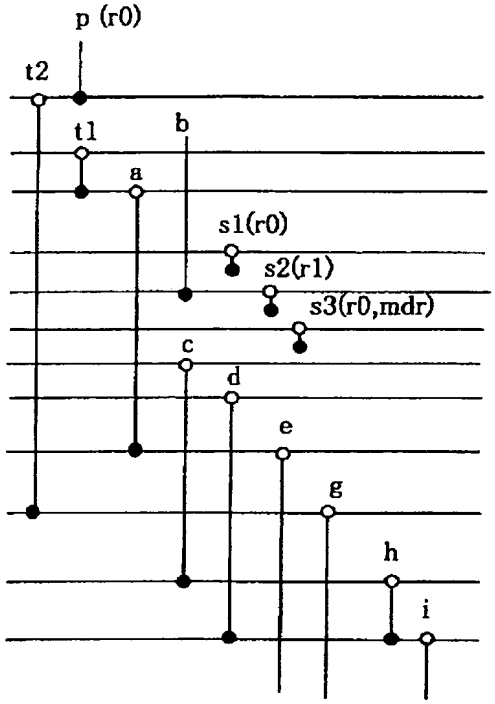
(d)

```
int f (int p){
    int a,b,c,d,e;
    ..........
    dm(a,b,c,d);
    ..........
    g = p + 10;
    ..........
}
```

【図１４】

```
i20: t2 = p;
     .....
i1:  t1 = t2 + b;
i2:  a  = t1 + 10;
     .....
i3:  asm  mov a,r0  ;
i4:  asm  mov b,r1  ;
i5:  asm  div r1,r0 ;
i6:  asm  mov r0,c  ;
i7:  asm  mov mdr,d ;
     .....
i8:  e =  a + 20;
     .....
i9:  g =  t2 + 30;
     .....
i10: h =  c;
     .....
i11: i =  d + h;
     .....
```

(a)中間プログラム



(b)生存区間

【図１５】

(a)

```
int f (int p){
    int a,b,c,d;
    .....
     c = a / b;
     d = a % b;
    .....
}
```

(b)

```
.....
mov r2,r4
div r3,r4          (1) 除算
mov r2,r5
div r3,r5          (2) 剰余算
mov mdr,r5
.....
```

(c)

```
.....
mov r2,r4
div r3,r4
mov mdr,r5
.....
```

(d)

div rm,rn ... レジスタrnをレジスタrmで除算し、
除算結果をrnに格納し、剰余結果を
特殊レジスタmdrに格納する。